Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

EXERCICI PUNTUABLE TECPRO

20/05/2019 Grau en Enginyeria de Sistemes TIC

COGNOMS:

NOM: GRUP de LAB:

Exercici 1. Donada la següent funció recursiva, a) Justifiqueu el resultat de cadascun dels seus doctestos b) Pel *doctest2* i *doctest5*, justifiqueu quantes crides recursives s'efecutaran

```
def npi(theinput):
    """
    >>> npi("the dog that barked")
    #doctest1
    >>> npi(["the", "dog", "that", "barked" ])
    #doctest2
    >>> npi(["the", ["dog", "that", "barked"]])
    #doctest3
    >>> npi(["the", ["dog", ["that", "barked"]], ["was bad"]])
    #doctest4
    >>> npi(["the", ["dog", ["that", ["barked"]]]])
    #doctest5
    >>> npi([[[["the"], "dog"], "that"], "barked"])
    #doctest6
    """
    if type(theinput) != list:
        return 0
    nose = 0
    for item in theinput:
        nose = max(nose, npi(item))
    return 1 + nose
```

Exercici 2. Apartat a) Justifica el cost assimptòtic en cas pitjor dels següents mètodes sobre la classe Stack, sent n el nombre d'elements de la pila.

1. El mètode top()
2. El mètode pop()
3. El mètode push()

Apartat b) Justifica el cost assimptòtic en cas pitjor dels següents mètodes sobre la classe Queue, sent n el nombre d'elements de la cua.

1. El mètode put()
2. El mètode get()
3. El mètode remove()

Apartat c) Justifica el cost assimptòtic en cas pitjor d'un bon algorisme d'ordenació i exemplifica'n el funcionament donada la llista [3, -1, 2, 89, 14, 22, 33, 0]

Exercici 3. Donada la definició de la classe Stack que segueix i les crides a la funció replacing, se us demana que escriviu el resultat d'execució dels jocs de proves que segueixen.

```
class Stack:
     def __init__(self):
         self.items = []

     def isEmpty(self):
         return self.items == []

     def push(self, item):
         self.items.append(item)

     def pop(self):
         return self.items.pop()

     def peek(self):
         return self.items[len(self.items)-1]

     def size(self):
         return len(self.items)

def replacing(dades):
    prec = {}
    prec["*"] = 3
    prec["/"] = 3
    prec["+"] = 2
    prec["-"] = 2
    prec["("] = 1
    opStack = Stack()
    info = []
    tokenList = dades.split()

    for token in tokenList:
        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or token in "0123456789":
            info.append(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()
            while topToken != '(':
                info.append(topToken)
                topToken = opStack.pop()
        else:
            while (not opStack.isEmpty()) and \
               (prec[opStack.peek()] >= prec[token]):
                  info.append(opStack.pop())
            opStack.push(token)

    while not opStack.isEmpty():
        info.append(opStack.pop())
    return " ".join(info)

if __name__=='__main__':
    print(replacing("A * B + C * D"))
    print(replacing("( A + B ) * C - ( D - E ) * ( F + G )"))
```

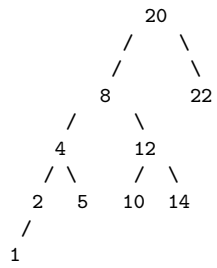
Exercici 4. Donada la classe —arbreBST—, corresponent a un Binary Search Tree, amb els mètodes implementats com segueixen,

```
class arbreBST(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None

    def insereix(self,k):
        if self.v==None:
            self.v=k
        else:
            if self.v==k:
                raise Exception("Repetit")
            elif self.v<k:
                if self.right is None:
                    self.right=arbreBST()
                self.right.insereix(k)
            else:
                if self.left is None:
                    self.left=arbreBST()
                self.left.insereix(k)
```

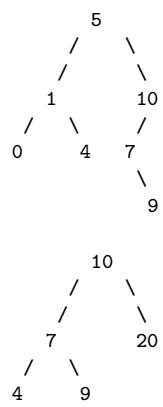
```
    def escriuFulles(self):
        #TO DO
    def chequejaArbres(self,other):
        #TO DO
```

Apartat a) Se us demana que implementeu el mètode recursiu *escriuFulles*, tal que, donat un arbre BST no balancejat, escrigui els nodes fulla de dreta a esquerra. Per exemple, en l'arbre següent, la resposta hauria de ser 22 14 10 5 1.




```
if __name__=='__main__':
    arbre=arbreBST()
    arbre.insereix(20)
    arbre.insereix(8)
    arbre.insereix(12)
    arbre.insereix(4)
    arbre.insereix(10)
    arbre.insereix(14)
    arbre.insereix(22)
    arbre.insereix(2)
    arbre.insereix(5)
    arbre.insereix(1)
    arbre.escriuFulles()
```

Apartat b) Escriu el mètode recursiu *chequejaArbres*, tal que, donats dos arbres BST no balancejats, escrigui els nodes en comú. Per exemple, pels arbres que segueixen, hauria d'escriure els valors 4 7 9 10.




```
a1=arbreBST()
a1.insereix(5)
a1.insereix(1)
a1.insereix(10)
a1.insereix(0)
a1.insereix(4)
a1.insereix(7)
a1.insereix(9)
a2=arbreBST()
a2.insereix(10)
a2.insereix(7)
a2.insereix(20)
a2.insereix(4)
a2.insereix(9)
print "chequeja", a1.chequejaArbres(a2)
```

Apartat c) Escriu quina és la complexitat en cas pitjor del mètode *escriuFulles* i quina és la complexitat en cas pitjor del mètode *chequejaArbres*. Utilitza la notació O, i justifica breument el seu cost.